Parallelization of the Tau-Leap Coarse-Grained Monte Carlo Method for Efficient and Accurate Simulations on GPUs

Lifan Xu[1], Stuart Collins[2], Dionisios G. Vlachos[2], and Michela Taufer[1]
[1]Global Computing Lab, Computer and Info. Sciences, University of Delaware
[2]Chemical Engineering, University of Delaware

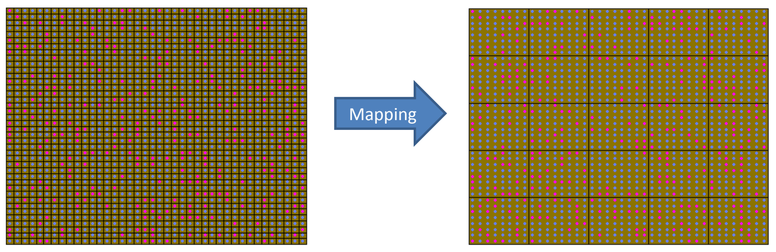
Abstract

Recent efforts have outlined the potential of GPUs for Monte Carlo scientific applications. In this poster, we contribute to this effort by exploring the GPU potential for the tau-leaping Coarse-Grained Monte Carlo (CGMC) method. CGMC simulations are important tools for studying phenomena such as catalysis and crystal growth. Existing parallelization of other CGMC method do not support very large molecular system simulations. Our new algorithm on GPUs provides scientists with a much faster way to study very large molecular systems (faster than on traditional HPC clusters) with the same accuracy.

The efficient parallelization of the tau-leaping method for GPUs includes the redesign of both the algorithm and its data structures to address the significantly different GPU memory organization and the GPU multi-thread programming paradigm. The poster describes the parallelization process and the algorithm performance, scalability, and accuracy. To our knowledge, this is the first implementation of this algorithm for GPUs.

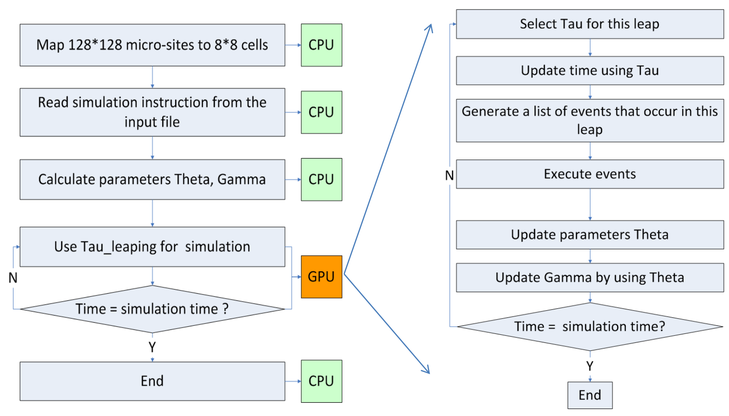
Coarse Grained Kinetic Monte Carlo Model (CGMC)

- Group neighboring microscopic sites together into "coarse-grained" cells
- Apply a closure at the stochastic level to resident molecules to describe their distribution in the cell



In the simplest closure above, molecules within cells are assumed to be well mixed. The molecules of each cell are allowed to interact with and diffuse to nearby cells.

CGMC Algorithm on GPUs

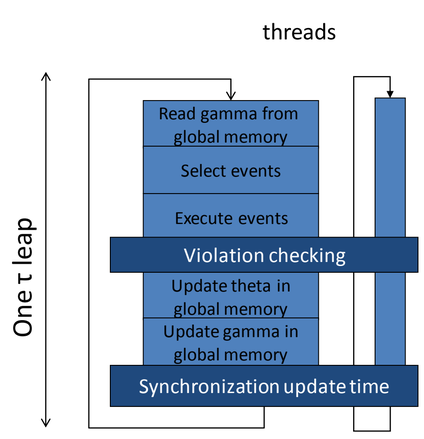


Implementation using global memory
- One thread simulates events in one cell
- All threads are executed in parallel
- Performance is much faster than CPU (**16 times**)

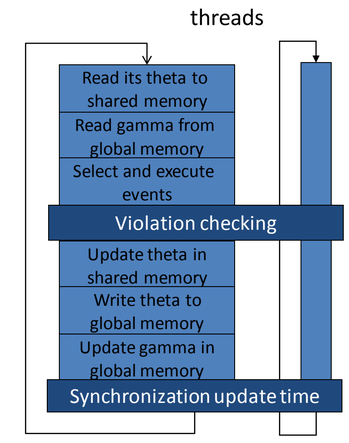
Improve the implementation to use shared memory (SM)
- Memory access cost in GPU:
  - 400-600 clock cycles to read/write a float from/to global memory
  - 4 clock cycles to read/write a float from/to shared memory
- Limitation for shared memory:
  - Has only 16KB per block
  - Can be only accessed by threads in a same block
- Improvement: make use of shared memory
  - Move theta to shared memory before leap starts
  - Write theta back to global memory after each leap
- Performance improves significantly **(100 times)**

Next steps …
- Move gamma to shared memory
  - Gamma relates to neighbors' theta
  - Cannot keep all threads in one block
- Endow threads more power to calculate their neighbors' theta

Multi-thread Implementation of CGMC



GPU implementation with global memory

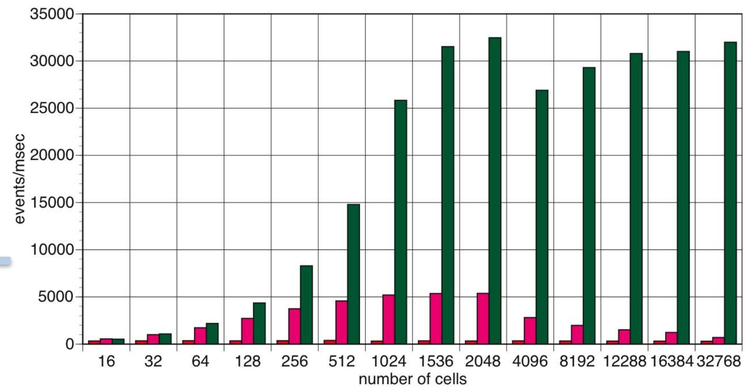


GPU implementation with shared memory

Performance

Platforms			
GPU -Tesla S1070		Intel(R) Xeon(R) CPU X5450(CPU)	
Number of cores	240 per GPU	Number of cores	4
Global memory	4GB	Memory	4GB
Clock rate	1.44 GHz	Clock rate	3 GHz

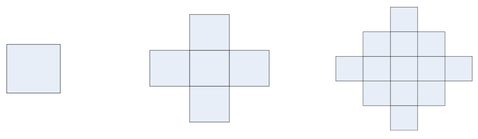
Global vs. Shared Memory



Re-thinking the Tau-leaping Algorithm for GPUs

Redesign of cell structures:
- Macro-cell = set of cells
- Number of cells per macro-cell is flexible

Redesign of event execution:
- One thread or one block simulates events in one macro-cell
- Events are replicated across macro-cells

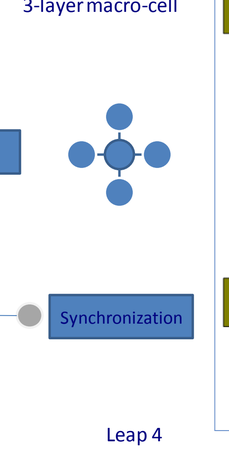


macro-cell 1-layer macro-cell 2-layer macro-cell 3-layer

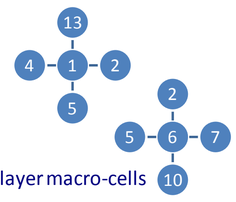
Coarse-grained parallelism implementation
- Use 2-layer algorithm
- One thread is in charge of one macro-cell
- No performance increase because threads execute 5 times more events in every first leap

Fine-grained parallelism implementation
- Use 2-layer algorithm
- One block is in charge of one macro-cell
- Each block has five threads corresponding to five cells in one macro-cell
- Performance significantly increase for small systems

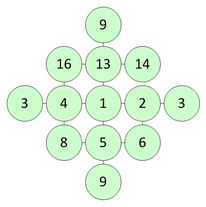
Next steps …
- Address limits in number of threads per GPU
- Use multiple GPUs for large systems



Cells in a 4x4 system

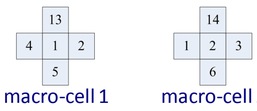


2-layer macro-cells



3-layer macro-cell

Event Replication

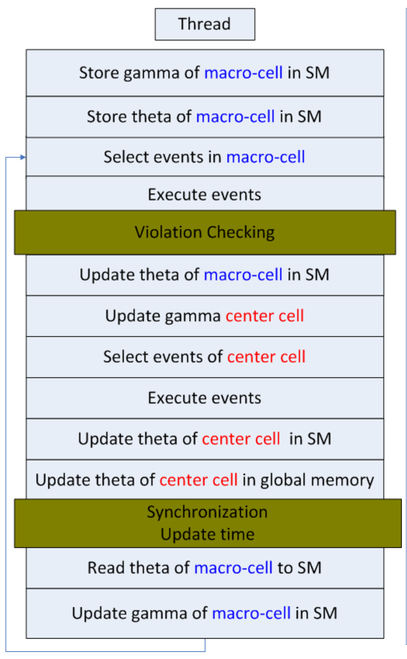


macro-cell 1 macro-cell 2

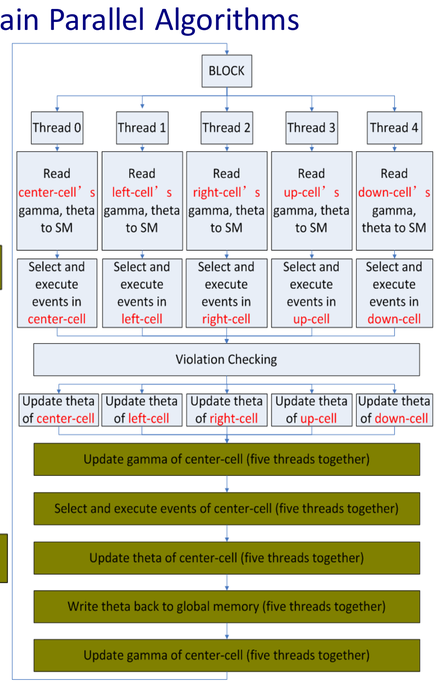
E1: A[cell 1] + B[cell 2] -> C[cell 1] + D[cell 2]
E2: A[cell 2] -> A[cell 3]
A,B, C, D are different molecule species.

	Macro-cell 1 (thread 0)			Macro-cell 2 (thread 1)			
	Cell 1	Cell 2	…	Cell 1	Cell 2	Cell 3	…
Leap 1	E1	E1		E1	E1	E2	
	E1	E1		E1	E1		
		E2			E2		
Leap 2	E1				E1		
					E2		
					E2		

Coarse vs. Fine Grain Parallel Algorithms

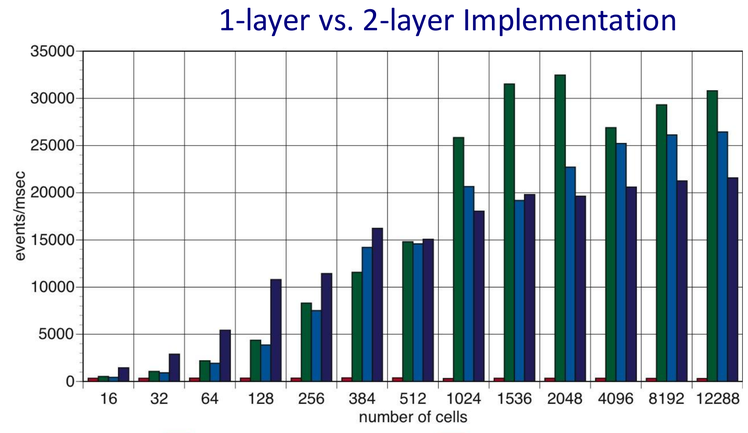


2-layer **coarse-grain** parallel algorithm

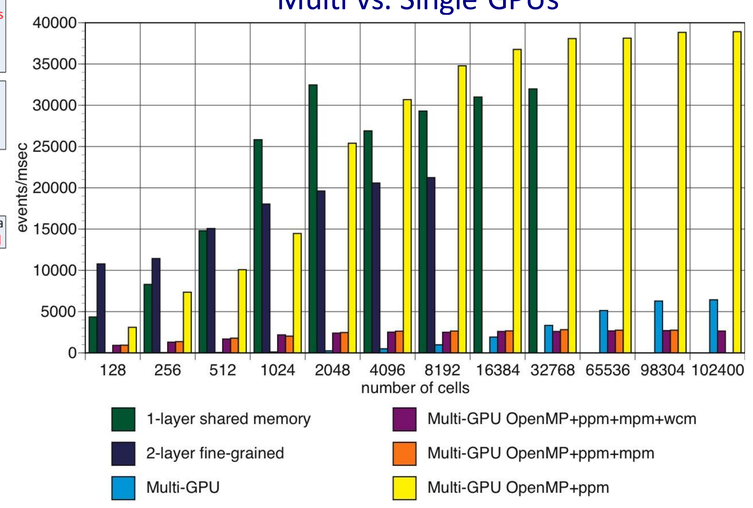


2-layer **fine-grain** parallel algorithm

1-layer vs. 2-layer Implementation



Multi vs. Single GPUs



Synchronization Frequency

Multi-GPU Implementation

Pseudo-code of multi-GPU implementation:
- CPU gets number of GPU and assigns simulation cells to GPUs
- CPU packs data and sends it to each GPU
- Each GPU copies data to its own memory and calls kernel function for two leaps
- All GPUs copy new data back to CPU
- CPU sends new data to GPUs and start next leap

Configuration:
- Multiple GPUs + OpenMP with portable pinned memory (ppm), mapped pinned memory (mpm), and write combined memory (wcm)
- Multiple GPUs + OpenMP with portable pinned memory (ppm), mapped pinned memory (mpm)
- Multiple GPUs + OpenMP with portable pinned memory (ppm)

Performance analysis:
- Number of events per msec is 120X faster than on a single CPU
- Max number of cells grows from 32,768 on single GPU to 102,400 on multiple GPUs

Conclusions and Future Work

Our contribution:
- Provide scientists with a much faster tau-leaping algorithm to study molecular systems with the same accuracy
- Prove that GPUs have tremendous computational horsepower
- Identify the most suitable level of parallelism for different molecular system sizes

Performance:
- Our tau-leaping algorithm is more than 100 times faster than the sequential version on CPU

Accuracy:
- GPU results are exactly the same as on CPU

Future Work:
- Improve performance of multi-GPUs with portable pinned memory
- Use other optimization techniques i.e., prefetching and memory coalescing

Acknowledgments

Accuracy

- In our testing simulation system, we have three molecule types: A, B, and C
- Initially we have 3000 molecules of type A in 16 cells
- In each leap, a certain number of events take place:
  - A can change to B and vice versa
  - B can react with B and change to C and vice versa
- We record number of molecules after each leap on CPU and GPU and compare the results
  - The results are exactly the same